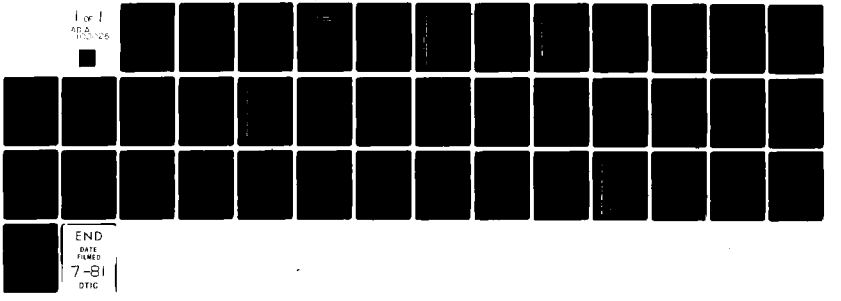
AD-A100 026 RAND CORP SANTA MONICA CA F/G 5/10
ADVICE-TAKING AND KNOWLEDGE REFINEMENT: AN ITERATIVE VIEW OF SK--ETC(U)
JUL 80 F HAYES-ROTH, P KLAHR, D J MOSTOW
UNCLASSIFIED RAND/P-6517 NL



ADVICE-TAKING AND KNOWLEDGE REFINEMENT: AN ITERATIVE VIEW OF SKILL ACQUISITION

Frederick Hayes-Roth, Philip Klahr, and David J. Mostow

July 1980

P-6517

ACKNOWLEDGMENTS

We gratefully acknowledge the substantive contributions of our Rand, Stanford, and Carnegie-Mellon colleagues to this work. John Burge collaborated with us on learning heuristics. Stan Rosenschein made frequent contributions to our efforts on knowledge representation. Doug Lenat assisted in the development of knowledge representations, heuristics, and procedures for cognitive economy (Lenat, Hayes-Roth, Klahr, 1979a, 1979b). As advisors to Jack Mostow, both Allen Newell and Jaime Carbonell contributed to our understanding of operationalization. The report, however, represents only the authors' attitudes and not the opinions of others with whom we have collaborated.

Perry Thorndyke made major contributions to the organization of this paper.

Several computer programs have been developed in the course of this research by various combinations of the authors or their collaborators. Mostow has implemented the knowledge programming system as part of his dissertation research supervised by Hayes-Roth. Hayes-Roth and Klahr have implemented three different programs for representing and applying the knowledge of the hearts domain. While these programs actually play the game moderately well, their interest for us lies in their alternative approaches to knowledge representation and control.

focus

WHAT IS LEARNED?

When we acquire an intellectual skill, what is learned? In this paper, we focus on three types of learned knowledge. The first includes improved concepts and heuristics that characterize the problem domain. The second includes increasingly effective means for achieving goals in the problem domain. The third type includes improved knowledge of the expectable consequences of our plans. In this context, skill development means improving the quality and increasing the coverage of these three kinds of knowledge as well as the capability to apply them appropriately.

Nearly all of our skills develop in part by instruction and in part by experience. We will elaborate this notion as a two-phase learning cycle. The first phase requires a learner to follow the advice of the instructor, as illustrated in Figure 1.

In the first phase, an instructor advises the learner about a problem domain. The advice specifies the basic concepts of the domain, rules that restrict behavior, and suggested heuristics to guide problem-solving behavior. Before the learner can employ this advice in the generation of behavior, he or she needs to convert it into useful forms. First, the learner needs to parse and interpret the advice. To understand each concept, rule and heuristic, the learner must relate it to his or her existing knowledge of the problem domain. We refer to this collection of knowledge as the learner's knowledge base. The learner's understanding transforms the advice from its initial linguistic form to a declarative knowledge representation. At this point, the

THE LEARNING CYCLE:
PHASE 1, FOLLOWING ADVICE

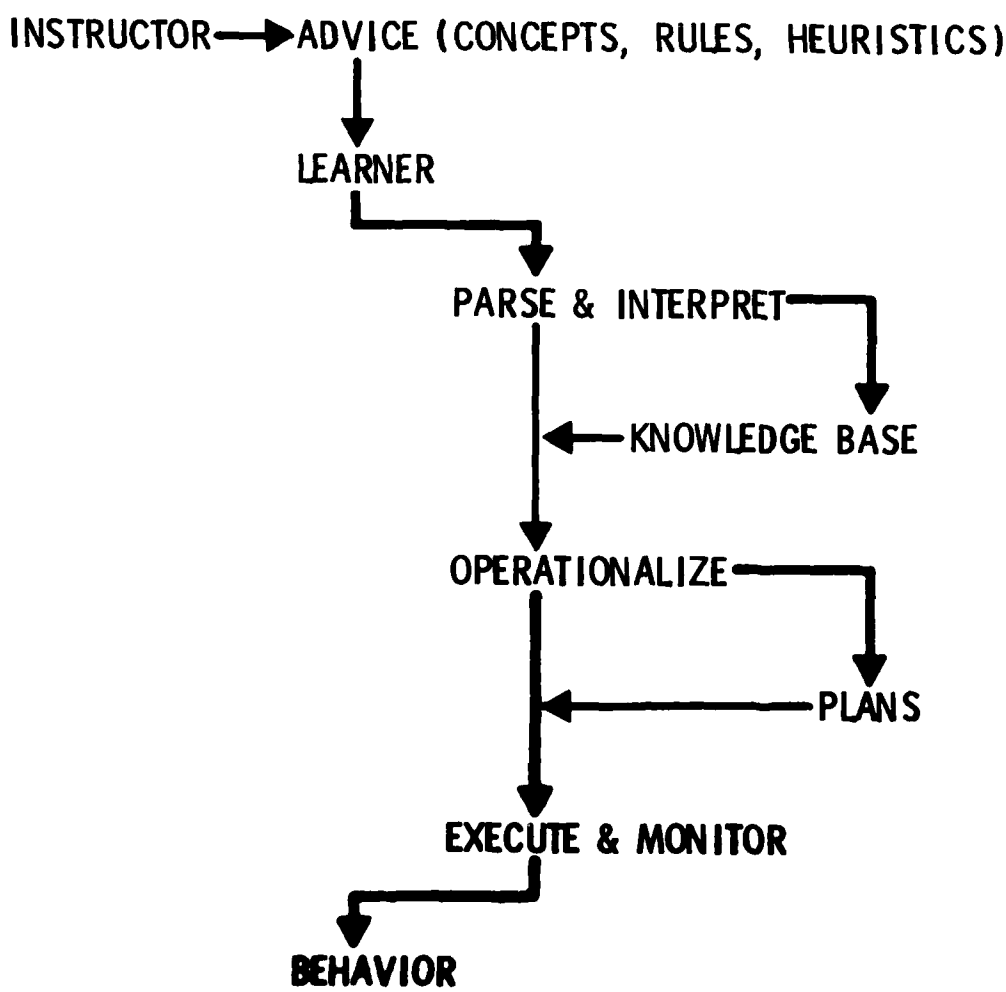


Figure 1

learner must operationalize the advice by transforming the declarative knowledge into executable or procedural forms. Operationalization means finding plans for carrying out the actions which the advice prescribes. Finally, the learner must execute and monitor the plans. This final step produces the behavior which the initial advice dictated.

As we all know, learning a new skill is rarely as simple or effective as the preceding discussion suggests. More often than not, our initial efforts at carrying out someone's advice lead to several problems. We may fail to carry out the instructor's true wishes because we misunderstood the advice. We may err because our plans fail to work as supposed. Or we may find it difficult to execute our plans as we hoped. When we begin to act, our behavior reveals such shortcomings. These weaknesses in behavior stimulate the second phase of the learning cycle.

In the second phase (Figure 2), we diagnose the problems in behavior and refine the knowledge that underlies them. These processes may be carried out either by the instructor or by the learner, or both. Diagnosis in this context means attributing responsibility for errors to units of knowledge which supported erroneous reasoning or procedures. Refinement means modifying previous knowledge to prevent similar negative experiences in the future. The refinement process may suggest new advice to eliminate ambiguities in the instruction, new concepts or heuristics for approaching problems, new plans for avoiding undesirable events, etc. In each case, the learner needs to reevaluate his knowledge base and plans in accordance with the diagnosed deficiencies and proposed refinements. Our principal objective in this paper is to explain each of these processes. We will largely ignore the role of

THE LEARNING CYCLE:
PHASE 2, DIAGNOSING BEHAVIOR AND REFINING KNOWLEDGE

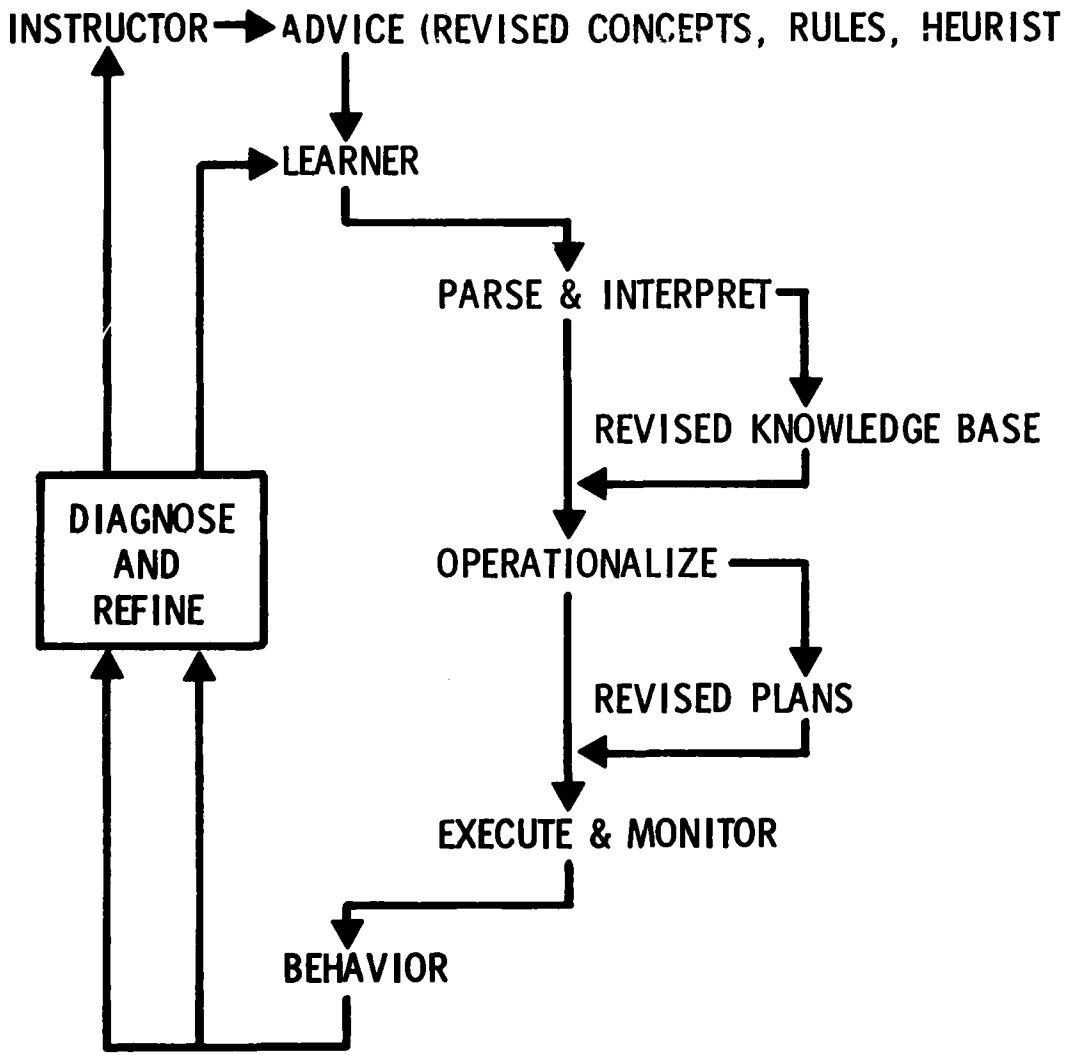


Figure 2

positive learning experiences whose diagnosis should lead to strengthening of responsible elements. However, some similarities between the treatments of positive and negative learning experiences are readily apparent. (See also Lenat, Hayes-Roth, & Klahr, 1979a, 1979b.)

To study the learning cycle, we have pursued a simple methodology, as shown in Figure 3. An instructor provides advice of the sort previously described. In the given problem domain, the advice covers the concept definitions, the behavioral constraints or rules, and some specific performance heuristics that prescribe particular methods for achieving goals. To model the advice-taking process, we are developing a knowledge programmer, an Artificial Intelligence (AI) system that assimilates such advice and converts it into an executable program for performing the instructed skill. In this conversion process, the knowledge programmer should produce a knowledge base that supports expectations about what its program will accomplish in various situations. When the program is allowed to perform in test situations, its behaviors will reveal how well the knowledge programmer has followed the advice.

To model the second phase of learning, we are developing AI systems for diagnosis and knowledge refinement. These systems must compare the actual program behaviors with the expected ones to diagnose weaknesses in the knowledge base. For each weakness, knowledge refinement strategies propose ways to modify the knowledge that will bring observed and predicted outcomes closer. The modified knowledge then must be fed back to the learner in the form of new advice, which reinitiates the learning cycle.

LEARNING RESEARCH METHODOLOGY

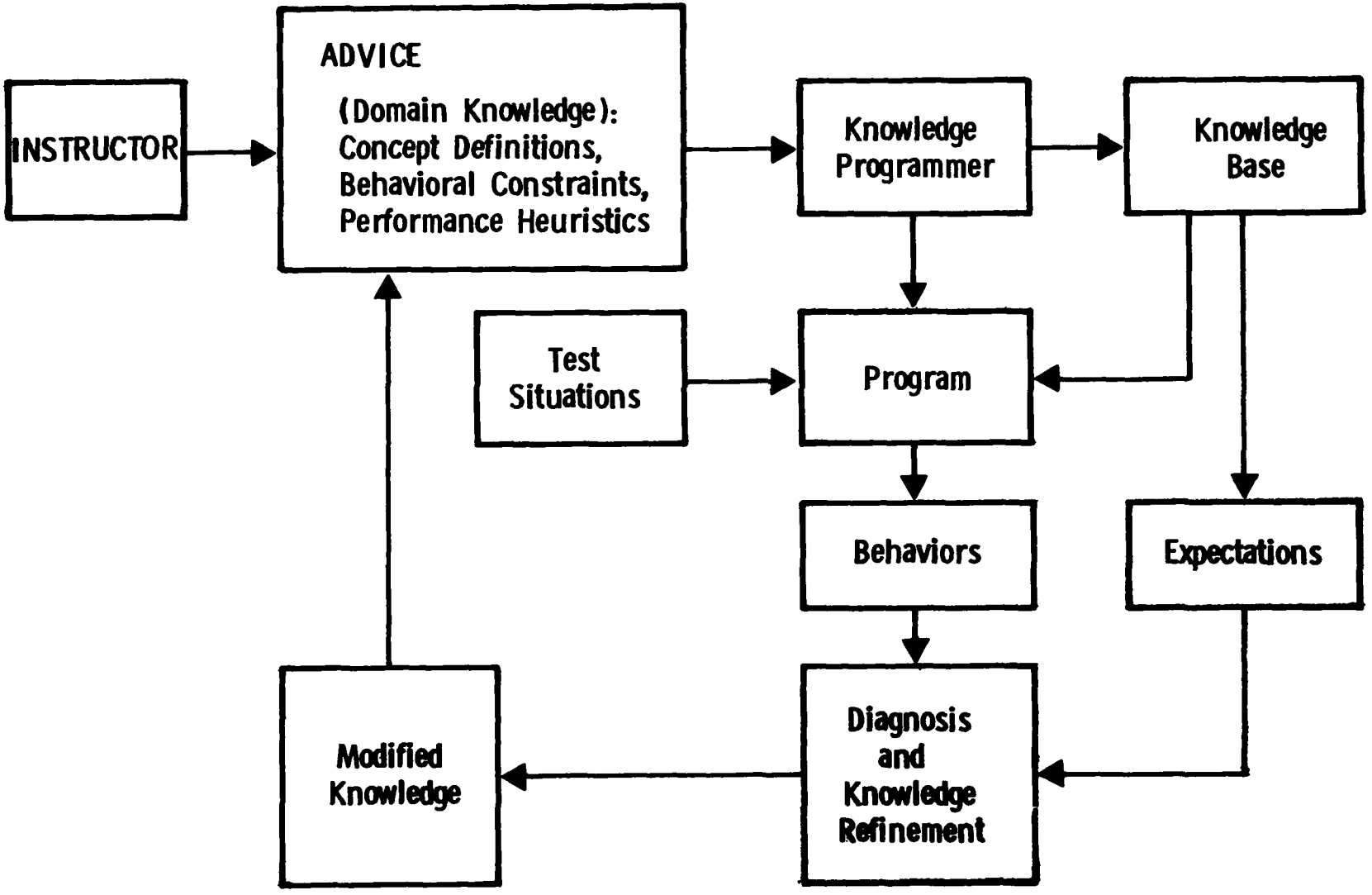


Figure 3

We have designed and implemented AI programs for a portion of each of these capabilities. Our systems are described in several technical papers (Hayes-Roth, Klahr, Burge, & Mostow, 1978; Hayes-Roth, 1980; Hayes-Roth, Klahr, & Mostow, 1980; Mostow & Hayes-Roth, 1979a, 1979b). In this paper, we will focus more on the "what" of these skill development processes than on the "how." Advice-taking and knowledge refinement seem to us ubiquitous aspects of skill development, and our research methodology emphasizes those aspects which require greatest advance. We will discuss our specific research applications where appropriate, but otherwise we will emphasize more general problems and the learning principles they suggest.

The proposed research methodology emphasizes four principal problems. First, we must specify what we mean by advice: What is advice and how shall we represent it? The second major problem concerns the conversion of advice into executable procedures: How can a learner operationalize advice? Third, we must specify the events that should initiate learning: What kinds of situations trigger learning? Fourth, we need to explain how to modify previously erroneous behaviors: What can be learned, and how can this be done? In the following sections, each of these problems will be considered in turn.

WHAT IS ADVICE?

Advice consists of concept definitions, behavioral constraints, and performance heuristics. Concept definitions describe the elements of a problem domain, such as objects, actions, and properties. Behavioral

constraints prescribe necessary qualities and proscribe disallowed characteristics of behavior. These constraints are typically mandatory rules that govern behavior in a problem domain. Performance heuristics, on the other hand, suggest ways to solve the domain problems and achieve desired results. These heuristics prescribe ways a learner should attempt to behave if possible.

Our research has been motivated, in part, by a desire to understand learning as a domain-independent capability. Nevertheless, we believe strongly that expertise in any domain requires considerable domain-specific knowledge. To bridge this gap, we have been developing general methods for assimilating domain-specific knowledge (as have Barr, Bennett, & Clancey, 1979; Davis, 1978, 1979, among others). In particular, we interpret domain-specific rules and heuristics as specific "compositions" of the domain concepts. From this perspective, advice consists of the primitive concepts of a domain plus the constraints and heuristics which are expressions composed in terms of these primitive concepts. The general capabilities that underlie this view include a scheme to represent concept definitions and a method to express rules and heuristics as composite expressions of the more primitive terms. In a sense, the domain concepts define a lexicon in which the rules and heuristics are expressed.

To illustrate, we will consider the familiar card game hearts. The following text would be representative of an instructor's advice to a learner.

> The goal of this game is to avoid taking points. In each round, the deck of cards is initially dealt out among 3 or 4 players. Each player's cards constitute the player's hand. Play in a round consists of a series of tricks. Each player, clockwise in turn, plays once in a trick by moving a card from his or her hand into the pot. The player who has the two of clubs (2C) leads the first trick. Each player must follow suit if possible, i.e. must play a card of the same suit as that led. The player who plays the highest card in the suit led wins the trick and leads the next trick. A player who wins a trick takes all cards in that pot and is charged with any points which those cards have. Each heart has one point (hence the name of the game). The queen of spades (QS) has 13 points. The worst a player can do in one round is take all but one point, in which case he or she takes 25 points. If the player takes 26 points, i.e. "shoots the moon," every other player is charged the 26 points. Shooting the moon is the best a player can do; taking zero points is second best. You can prevent someone from shooting the moon by taking a point yourself. Whenever it is out, flush the queen of spades to make sure it isn't given to you.

A complete description of the game can be found elsewhere (Balzer, 1966; Hayes-Roth, Klahr, Burge, & Mostow, 1978).

The preceding description illustrates all three kinds of advice elements. In the hearts domain, the concept definitions include the following:

deck, hand, card, suit, spades, clubs, diamonds, hearts,

deal, round, trick, avoid, point, player, play, take,

lead, win, follow suit, shooting the moon, avoid, ...

In terms of these concepts, the behavioral constraints include:

The player with 2C leads.

Each player must follow suit.

The player of the highest card in the suit led wins the trick.

The winner of a trick leads the next trick.

The performance heuristics include:

Avoid taking points.

Take all the points in a round.

If you can't take all the points in a round, take as few as possible.

Take a point if necessary to prevent someone from taking them all.

If the queen of spades is out, flush the queen of spades.

Now let's consider how these domain concepts can be represented and used to understand the domain-specific advice. In particular, consider the concepts of "trick" and "avoid." A "trick" is defined as a scenario in which each player plays a card and the winner takes the pot. Similarly, we define "avoid" by saying that avoiding an event during a scenario means not letting it happen then. In our work on knowledge programming, we have adopted specialized forms of the lambda calculus to represent such concepts (Allen, 1978). In our representation scheme, we would represent these domain concepts as follows:

```
(trick) = (scenario (each p (players) (play-card p))
                    (take-trick (trick-winner)))

(avoid e s) = (achieve (not (during s e)))
```

These representative definitions highlight our methods. First, each concept is viewed as a LISP expression, and the equality means that the expression on the right can be substituted for the one on the left. Each domain-specific concept is represented using a lexicon of

approximately 100 domain-independent primitive concepts, such as "scenario," "each," "achieve," "not," and "during." Thus, the definition of "trick" above expresses exactly:

> A trick is a scenario of two sequential steps.
> First, for each player p in the set of players, p
> plays a card
> Second, the trick-winner takes the trick

In this definition, "play-card," "take-trick," and "trick-winner" are stand-ins for other domain concepts that are similarly represented in terms of lower-level definitions. Play-card, for example, is a predicate that denotes the action of playing a card; similarly, take-trick denotes winning the trick and taking the pot.

The definition of "avoid" expresses the notion that some event e is to be avoided in some scenario s. In particular, it states that avoid e in s is equivalent to achieving the negation of the proposition that e occurs during s.

In our research, we have studied advice-taking in several domains, including hearts and music. The same core set of primitive concepts is used to define the domain concepts. In hearts, all of the basic domain concepts are represented as LISP expressions like those shown above. Moreover, each bit of behavioral advice is represented as a LISP expression as well. However, simply representing advice in the form of such expressions is not sufficient to follow the advice; the expressions must be executable. Making such expressions executable is the goal of operationalization.

HOW CAN A LEARNER OPERATIONALIZE ADVICE?

Even if a learner can map behavioral advice into LISP expressions or other representation formalisms, he or she may have no effective means of carrying out the actions needed to achieve what the expression denotes. In this framework, however, operationalization has a simple meaning: Expressions representing behavioral prescriptions must be transformed into operational forms (Balzer, Goldman, & Wile, 1977; Barstow, 1977; Heidorn, 1974). In our context, an expression of the form $(F\ e_1\ \ldots\ e_2)$ is operational only if F has an effective procedure and each e_1 is operational or denotes a constant. While this may sound quite abstract, it really is quite simple: A prescribed behavior is operational if we can express it in terms of procedures and objects we know how to evaluate.

Again, a specific example will help clarify these points. Suppose, for example, a learner wants to follow the heuristic, "avoid taking points." The learner might reason as follows:

(1) To avoid taking points in general, I should find a way to avoid taking points in the current trick.

(2) This means I should try not to win the trick if it has points.

(3) I can do this by trying not to play the winning card.

(4) This I could do by playing a card lower than some other card played in the suit led.

(5) Thus, I should play a low card.

Each successive statement in this line of reasoning represents a specific way to achieve the effect described in the preceding one. The last statement, however, is the only one which fits our definition of operational. Statement (5) represents an effective operationalization of the initial advice. In our opinion, "learning" often depends primarily on the learner's ability to find ways for carrying out an instructor's advice. As this example shows, a "problem-reduction" strategy may prove sufficient to convert an unoperational goal expression into an effective, sufficient, operational procedure.

This example may seem to suggest that operationalization may be quite straightforward and simple. We have found, however, that it requires substantial analytical techniques that may give rise to considerable complexity. To convert one expression into its next-level representation, the learner can use a variety of methods. Basically, the learner substitutes a new expression which he or she thinks will suffice to achieve the previous one. Three different types of substitutions can arise. Each substitution may replace an initial expression by a logically equivalent one, by a description of some state whose realization should entail the initial one, or by a set of actions that supposedly can produce the desired result. Two kinds of knowledge are required to accomplish these transformations. First, the learner manipulates all of the domain-specific knowledge to determine various relations of interest. Second, the learner must also apply general transformational rules that determine plausible and correct substitutions. We have, to this date, developed about 200 of these transformational rules.

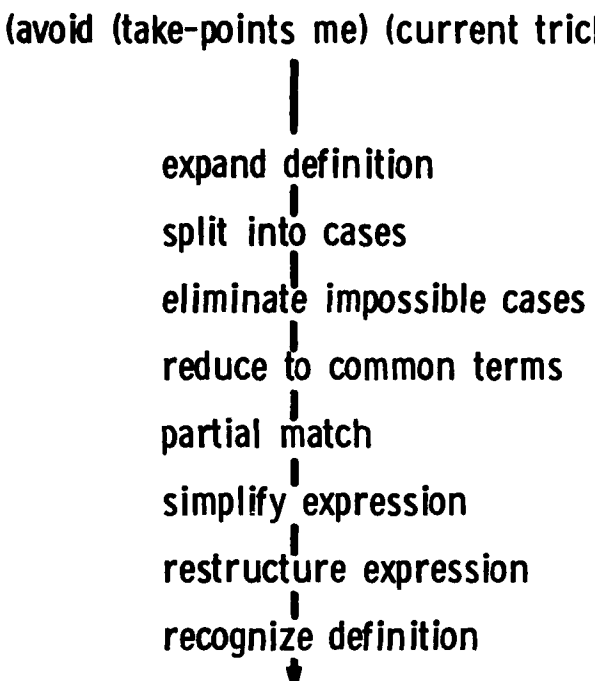
Figure 4 illustrates at a fairly high level the successive expressions that result from the knowledge programmer's efforts to operationalize the "avoid taking points" heuristic. On the left, five LISP expressions are shown in correspondence with the five steps a typical human learner takes. Between the first and second expression, we have shown eight distinct types of transformations that play a role in transforming expression (1) into expression (2). While a detailed explanation of each transformation would exceed the scope of this paper, we will describe each of these kinds of operations briefly below.

The first kind of transformation expands the definition of a term in an expression. For example, the initial expression would be matched to the form (avoid e s) so that (take-points me) would replace e and (current trick) would replace s. The resulting expression would be an expanded expression for the idea that the learner should prevent taking points during the current trick:

(achieve (not (during (current trick) (take-points me)))).

This kind of definitional transformation simply reexpresses a compact idea in terms of its equivalent and more elaborated elementary concepts. Definition expansion plays a major role in operationalization because a learner must frequently unpack and manipulate advice that is initially packed in terse compositions of domain concepts. By similarly expanding the concept of trick, the previous operationalization becomes:

OPERATIONALIZE "AVOID TAKING POINTS"



(achieve (not (and [= (trick-winner) me] [trick-has-points]))))	Try not to win the trick if it has points.
(achieve (not (= (card-of-me) (winning-card))))	Try not to play the winning card.
(achieve (exists c (cards-played-in-suit) (lower (card-of-me) c)))	Make my card lower than some other card played in suit.
(achieve (low (card-of me)))	Make my card low.

Figure 4

```
(achieve (not (during (scenario
                        (each p (players) (play-card p))
                        (take-trick (trick-winner)))
                      (take-points me)))).
```

The second kind of transformation, called case analysis, breaks a single expression into two or more expressions which depend on alternative additional assumptions. In operationalization, case analysis reformulates one meaning as different expressions that can be evaluated separately. In the context of our current example, to insure against taking points during the two-step scenario, the learner might consider two cases: (1) that taking points occurs during the playing of cards or (2) that taking points occurs during the taking of the trick. This transformation would produce the following new expression:

```
(achieve (not (or
                (during (each p (players) (play-card p))
                        (take-points me))
                (during (take-trick (trick-winner))
                        (take-points me)))))
```

The third type of transformation eliminates impossible cases. The learner must recognize that some expressions cannot be achieved, and thus should be eliminated from further consideration. In the current example, case (1) should be ignored because there is no co-temporaneous relation between playing cards and taking points. More generally, impossible cases arise when expressions represent unachievable conditions.

Our knowledge programmer detects these in various ways. In this particular example, because it can find no relationship between the definitions of playing cards and taking points, it decides to drop that case from consideration.

The fourth kind of transformation reduces different expressions to common terms. This general transformation helps coerce comparability between seemingly different entities. In the current example, such a transformation helps reexpress case (2) above. This case represents the possibility that taking points would occur during the action of taking the trick. This case follows from our definitions of the game, because a person takes points when he or she takes a pot containing point cards. The knowledge programmer recognizes this relationship as a path in a semantic network it constructs to represent part-whole relationships among domain action definitions. Because take-points is a part of take-trick, the knowledge programmer can reexpress (during (take-trick (trick-winner)) (take-points me)) as follows:

```
(exists c1 (cards-played)
   (exists c2 (point-cards)
      (during (take (trick-winner) c1)
              (take me c2)))).
```

This expression characterizes the meaning of "me taking points during the trick-winner's taking a trick." This can only make sense, according to the domain concepts, if me takes a point card c2 during the period that the trick-winner takes the cards played. Our transformation

has introduced two variables, c1 and c2, which remain to be made operational before the operationalization is complete.

The fifth general transformation method used is partial-matching (Hayes-Roth, 1978). Partial-matching compares two descriptions to identify their similarities and differences. When two expressions occurring within some relationship match perfectly, the relationship between them often can be greatly simplified. When this isn't possible, we can often achieve comparable simplifications by first finding ways to reduce the differences between the two expressions to common terms. By using a rule of partial-matching plus a semantic network of relationships among concepts, our knowledge programmer transforms the expression:

(during (take (trick-winner) c1) (take me c2))

into the requirement:

(and (= (trick-winner) me) (= c1 c2))

This requirement establishes necessary conditions for me to take points while the trick winner was taking the cards played, namely that the trick winner is me, and that me takes the point cards that the trick winner does.

The sixth type of transformation is simplification. Much general analytical knowledge concerns the types of expressions that can be replaced by simpler ones. DeMorgan's laws in logic are examples of the kinds of simplifications a learner implicitly uses to replace a complex expression by an equivalent, simpler one. Other kinds of simplifications include replacing symbolic expressions by constants or removing

quantifiers when appropriate. In the current example, this last kind of simplification can exploit the fact that the two variables c1 and c2 are equal. To simplify the expression, the symbol c2 can be replaced by c1, and at the same time the condition that c1 and c2 should be equal can simply be dropped.

The seventh type of transformation restructures expressions to make them more amenable to the other types of analysis. These restructuring operations only serve the purpose of reconfiguring expressions, but such reconfigurations can make the difference between a hard and an easy operationalization. Depending on the domain knowledge and general transformations available, some expression patterns will sustain continued operationalization while others, perhaps equivalent, will not be noticed. In the current example, the learner profitably moves the existential quantifier to an interior position in the expression to restructure part of the expression as shown below:

```
(and [= (trick-winner) me]
     [exists c1 (cards-played)
         (in c1 (point-cards))])
```

This expresses the requirement that the trick winner is me and some c1 among the cards played is in the set of point cards.

The eighth type of transformation is the inverse of the first. These transformations recognize instances of higher-level definitions and replace the more complex, lower-level expressions by the corresponding higher-level ones. The resulting expressions may simplify otherwise complex expressions and may lead more directly to the identification of

effective procedures. In the current illustration, for example, the last two lines of the preceding expression represent exactly the concept that "the trick has points." This concept occurs in the second line of Figure 4's derivation as [trick-has-points]. By substituting this concept, the overall derivation to this point becomes:

```
(achieve (not (and [= (trick-winner) me]
                   [trick-has-points]))).
```

In operationalizing the remaining three steps, several other general types of transformations are involved. Approximately forty transformational steps are required to operationalize this single heuristic bit of advice. Other lines of reasoning could also be pursued successfully and several of these produce alternative operationalizations for "avoid taking points."

The one example we have followed should illustrate both the variety of techniques involved in operationalization and the complexity of the detailed processes which the knowledge programmer models. This particular example was chosen because of the variety of operationalization methods it reveals.

A second example of operationalization will provide the basis for illustrating the second phase of learning, involving diagnosis and refinement. In this example, the learner attempts to follow the instruction, "If the queen of spades is out, try to flush it." To follow this advice, the learner must develop a method for (1) deciding if the queen of spades is out and (2) forcing the player who has it to play it. Our knowledge programmer uses a "pigeon-hole principle" to solve

the first problem: A thing must reside momentarily in just one of its possible loci. In this example, the card is "out" if it has not been played and is not in the learner's own hand.

To flush the queen, a more complex operationalization effort is required. In overview, the knowledge programmer develops the following plan:

(1) Flush the queen of spades.

(2) This means forcing the owner of the queen of spades to play it.

(3) This requires me to reduce the owner's options to just that one action.

(4) This can be done by leading spades until the owner of the queen is forced to play it.

(5) To do this, I should win the lead and then continue leading spades.

The kind of reasoning the program uses to develop its plan is as follows: By substituting the definition of "flush," it infers that it needs to establish the condition "some player p must play the queen of spades." It uses its concept of "must" and the rule that players must follow suit to infer that p can have only one legal card to play--the queen of spades. This in turn entails either (1) p has only the queen of spades or (2) me leads a spade, and p's only spade is the queen. It focuses on the second case and then develops a plan for how player me could force such a situation. In brief, it develops a plan for me to win a trick to take the lead. Then as long as me retains the lead, me continues to lead spades. As players familiar with the game will realize, this is an effective method for flushing the queen. However, as we

discuss below, this plan will probably lead to some disappointments that should stimulate learning.

WHAT KINDS OF EVENTS TRIGGER LEARNING?

We presume that learning is often triggered by violated expectations. Expectations vary from weak, general ones to strong, specific ones. The weak expectations a learner may have arise from the general belief that following advice should produce favorable results. Equivalently, the learner may expect only good consequences when he or she follows the plans which operationalize an instructor's advice. Strong expectations on the other hand are consequences of actions that can be deduced from the learner's knowledge base. In particular, the learner operationalizes expressions to achieve particular results. Presumably then, executing the operationalizations should achieve the conditions they purport to.

Violated expectations are the triggering events for learning. These violations include both unexpected and unfavorable consequences of successfully executed plans. In addition, the unexpected but desirable outcomes of behaviors should trigger efforts to learn what these outcomes might be attributed to. Finally, disconfirmations of assumptions also motivate learning (Lakatos, 1976). In toto, we can think of behavior as theory-driven, because behaviors derive from assumptions, plans, and theoretically appropriate transformations of expressions; learning is triggered by events that disconfirm the expectable consequences

of such theories. A deep and more formal exposition of theory-driven learning appears in Hayes-Roth (1980).

A simple example will illustrate these concepts. Consider the previously derived plan for "flush the queen of spades." The plan developed for this advice was, roughly, take the lead, then lead spades until a player is forced to play the queen. Suppose that this plan worked well when executed in several games, but then during one game a sequence like the following unexpectedly occurred: The learner wins a trick. He or she then leads the jack of spades, and the other players follow suit. The queen is still held by one of the players. On the next trick, the learner chooses another spade to lead. This time, he or she has only two spades left, the four and the king, and chooses arbitrarily to play the king. The next player plays the five, the one after him plays the queen, and the last plays the ten. The learner has just won a trick according to its plan, and has even flushed the queen. Unfortunately, the learner has also taken 13 points, a very undesirable outcome.

What might the learner do at this point? With apparently little effort, a person would recognize that the plan was buggy, because it achieved an undesirable result that was unexpected. Implicit in the plan was the notion that the player with the queen would be coerced into playing it and, presumably, would win the spade trick with it. Having discovered this insight, a human learner would attempt to amend the buggy plan accordingly. The fix in this case would require that, when trying to flush the queen, a player must lead only spades below the

queen to avoid winning the queen. In the next section, we describe learning techniques that produce this sort of knowledge refinement.

WHAT CAN BE LEARNED, AND HOW IS THIS DONE?

A learner can increase its knowledge in any of the areas knowledge is currently possessed. This means either learning new domain knowledge, new operationalizations, or new operationalization methods. We will consider only the first two of these types of learning, because the last surpasses our current understanding. Thus, a learner can discover new and generalized domain concepts (Lenat, 1976; Lenat & Harris, 1978), additional behavioral goals, and improved heuristics for problem-solving in the domain. Each newly formulated unit of knowledge domain typically corresponds to unoperational advice. On the other hand, the learner can formulate new and improved plans for more effectively operationalizing previous advice.

We view learning of this sort as knowledge refinement. The learner progresses by iteratively modifying the current knowledge base to accommodate experience. The improvements to knowledge arise from two principal sources: heuristic rules for diagnosis and heuristic rules for learning. The diagnosis rules identify problematic knowledge base elements (cf. Sussman & Stallman, 1975). The learning rules suggest particular knowledge refinements. These elements support steps 3 and 4 of an overall five-step process, as shown in Table 1.

Let us pursue the example from the preceding section. We supposed that a learner would attempt to fix the apparently buggy plan for "flush

Table 1

KNOWLEDGE REFINEMENT APPROACH

Step	Source or Mechanism
1. Establish expectations	During knowledge programming, planning establishes plausible antecedents and consequences of actions; these beliefs represent expectations.
2. Trigger analysis	When an actual event violates an expectation, the reasoning behind the expectation is reanalyzed in light of observable data.
3. Locate faulty rules	A set of diagnostic rules debugs the planning logic by contrasting the a priori beliefs with actual data. If a heuristic rule used by the plan assumes a false premise or entails a false conclusion, it is faulty.
4. Modify faulty rules	A set of learning rules suggests plausible fixes to the erroneous heuristic rule. These might alter its preconditions, assumptions or expectations to keep it from producing the same faulty result in a subsequent situation.
5. Reimplement and test	Incorporate a modified heuristic rule into a new behavior by reinvoking the knowledge programmer. Verify that the rule eliminates the previous problem and test it in new situations.

the queen of spades" by asserting an additional goal for that plan to achieve. That additional goal, or constraint, was that the learner should not itself win the trick in which the queen of spades is played. That kind of advice could arise externally, from an instructor, or internally, from the learner's own diagnosis procedures. We will explore this latter alternative further.

First let us suppose that the learner had no precise expectation regarding the queen-flushing plan. However, since the learner followed supposedly expert advice, he or she has a general expectation that bad consequences should not result. When, as in this case, undesirable results occur, the learner tries to understand why he or she suffered such a regrettable outcome and how to prevent it.

The learner analyzes the last trick to infer cause-effect relations, based on its current knowledge. According to the domain concepts and constraints, to take 13 points in the trick, the learner had to win the trick during which the queen was played. So the learner conjectures some refined advice for itself: "Flush the queen of spades but do not win a trick in which the queen is played." Because this refined advice surpasses the original advice in quality, the learner has already improved its knowledge. On the other hand, this high-level advice requires operationalization if it is to be useful. This type of advice can be operationalized with exactly the same methods we previously described. Our current knowledge programmer correctly operationalizes this advice by producing a plan corresponding to the following: "Take the lead; then continue leading spades which rank below the queen." Thus, this type of bug could be eliminated by formulating a desired

refinement directly in terms of a new high-level prescriptive heuristic. The refined heuristic, in turn, is implemented by the same knowledge programming methods previously used for accepting advice from an instructor. (In some cases, as in this example, the refined heuristic can also be operationalized directly by modifying the previous plan, as opposed to starting over from scratch.)

To continue the illustration, let us suppose that the learner begins to apply its refined plan. Because the learner knows that the plan has been refined to prevent it from taking the queen of spades itself, it notes this specific expectation in the knowledge base as a predicted consequence of the plan. In a new game, however, suppose the learner has the ten, jack and king of spades. It wins a trick, then leads the ten. All players follow suit with lower cards, so the learner leads again with the jack. Again it wins the trick. At this point, its revised plan proscribes leading spades, so it leads a diamond. Another player wins the trick, and that player continues to lead spades. The learner is forced to play the king, and the next player follows suit with the queen of spades.

Again, contrary to its specific expectation, the learner wins the trick and takes 13 points. Now the learner attempts to discover why its expectation was violated. It constructs a cause-effect model of the events leading to the disaster. In this model, it notes that at the time it played the king, it had no other choices. So apparently, by that time, only by keeping the other player from leading spades could it have prevented the disaster. Alternatively, it reviews events prior to that trick to see what, if anything, it did that causally contributed to

creating a situation in which no options existed. It notices that playing the ten and jack of spades earlier produced the state where it had only the king of spades. It notes that these actions were taken with the express intention of preventing it from taking the queen, but apparently they contributed directly to just that outcome.

The learner now proposes to itself another refinement. It should prevent a reoccurrence of this type of situation in the future. Its proposed advice: "Do not lead low spades if you can be forced subsequently to play a spade higher than the queen."[1] This, in turn, leads to an operationalization that requires an estimation of the probable distributions of spades among players. While we have developed some methods for handling such probability functions, we have not yet implemented those needed for this particular problem. However, as persons knowledgeable in the game will notice, the proposed concept of a card that is "safe" vis-a-vis the opposing distributions is quite sophisticated. In fact, generalizations of this "safe spade" concept, such as "safe in suit x" or "safe with respect to all suits," play major roles in expert strategies.

---

[1] This example has not actually been performed by a machine implementation. Before it could be implemented, several difficult issues would arise. Foremost among these, the diagnostic system would need to conjecture several alternative problems and solutions. Each of these proposed solutions would require, in turn, experimental testing through additional play. For example, the program might have hypothesized the remedy, "Do not begin to flush the queen of spades if you cannot retain the lead." This heuristic seems beneficial, but we cannot be certain. Empirical validation of alternative heuristics seems unavoidable.

As another example of knowledge refinement, consider again the plan developed in a previous section to avoid taking points. That plan proposed playing the lowest possible card. Using this plan, the learner expects it will avoid taking points, but there are numerous ways that this plan leads to violated expectations, each of which reflects characteristics similar to the queen-flushing examples. For example, the learner may play its lowest card (a five, say) and still win a trick with points. This causes the learner to weaken its expectations (i.e., to associate some uncertainty with this predicted outcome). Pursuant to such a play, it may take another trick with its current lowest card (a ten, say), again with points. However, if it had played the ten before the five, it might have avoided winning the second trick, because in the second trick the five might have been lower than other players' remaining cards. Each of these problems gives rise to new attempts to refine both the expectations and the plan, in a manner similar to that previously described.

Our general model of knowledge refinement can be portrayed as shown in Figure 5.

The contrast between expectations and actual outcomes serves to focus the learning system directly on specific problems. The learner then attempts to diagnose the flaws in its original causal model in light of the new data in hand. The diagnosed problem dictates additional conditions or new goals for operationalization.

The overall approach we have taken to this knowledge refinement employs three basic elements: (1) proofs, (2) diagnostic rules, and (3) learning rules. While not actually implemented on a computer, we

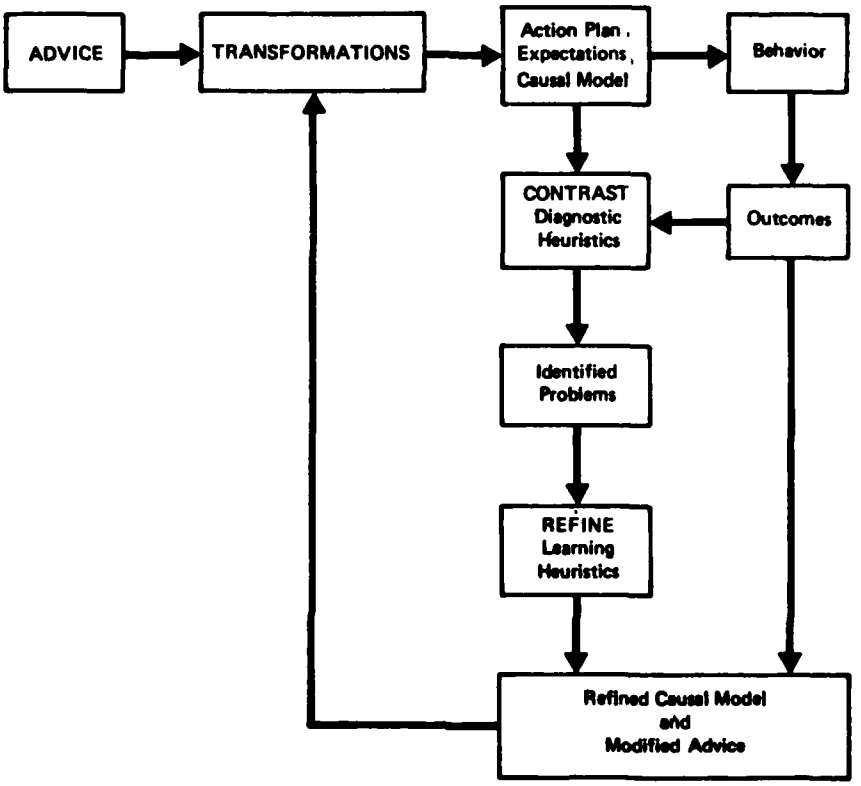


Figure 5--Knowledge-refinement strategy

have hand-simulated all of the steps. The knowledge programmer associates with each plan and its expectations a proof (or an informal rationale). The proof of a plan links assumed conditions to expectations by deductively following paths representing the transformations used to operationalize the plan. Links along these paths reflect the equivalence of logical transformations, the plausible sufficiency of heuristic transformations, or the antecedent-consequent relations of incorporated instrumental acts. At each point, a transformation links premises to expectations, and these expectations may become part of the premises for a later inference. In short, a proof maps a general model of cause-effect relations into a specific derivation of the expected consequences of particular planned actions.

Diagnostic rules examine the proof in light of the evidence and identify hypothetical deficiencies in the knowledge base. A typical diagnostic rule is as follows:

Invalid Premise Diagnostic Rule

> If an expectation is violated, find a premise in the proof of the expectation that is falsified by the data. If the false premise follows from some inference rule whose own antecedent premises (necessary conditions) are true, declare that rule faulty.

Learning rules, on the other hand, specify ways to modify heuristics to correct deficiencies. We have generated a sizable set of such rules to date. Two examples of learning rules are shown below:

Require Implicitly Assumed Premise Explicitly

> If an implicit assumption of a rule is falsified during proof analysis, add the premise to the required conditions of the rule and delete any other premises that it implies.

Guarantee Assumed Conditions

> If an assumed premise is falsified during proof analysis, identify sufficient conditions for its validity and make these required conditions for the associated plan component.

Figure 6 demonstrates how these diagnostic and learning rules are used to refine the original "flush the queen of spades" plan as discussed above. Figure 6 also exemplifies the knowledge refinement approach outlined in Figure 5.

Thus, we have found several ways to evaluate a plan against observable outcomes to identify weaknesses, conjecture refinements, and evaluate these refinements experimentally. Very little of this work has been implemented in applications software, because of the large number of plausible learning strategies (see Hayes-Roth, Klahr, & Mostow, 1980) and the wide variety of specific possible applications. Any efforts to implement these concepts in a realistically complex task will encounter considerable combinatorial difficulties. Each observed error may indicate several hypothetical bugs and fixes. Each of these will require independent empirical (or formal) validation, usually accomplished best by experimental testing. The need for testing hypothetical concepts and

PLAN:
Flush queen of spades:

If player P takes the lead
and P doesn't have the queen of spades
then P continues leading spades

Expectation:
P doesn't take the queen of spades

Proof of expectation:

1. Player P takes the lead	Premise (condition of plan)
2. P doesn't have the queen of spades	Premise (condition of plan)
3. P continues leading spades	Premise (action of plan)
4. If player P takes the lead and P doesn't have the queen of spades and P continues leading spades then opponent will play queen of spades.	Heuristic rule
5. Opponent will play queen of spades	Derived premise from 1,2,3,4
6. If an opponent plays queen of spades then the opponent wins the trick and the opponent takes the queen of spades.	Heuristic rule
7. Opponent takes the queen of spaces	Derived premise from 5,6
8. If opponent takes the queen of spades then player P doesn't take queen of spades.	Heuristic rule
9. Player P doesn't take queen of spades	Derived premise from 7,8

Behavior in actual play: P leads king of spades;
Opponent plays queen of spades.

Outcome: P wins the trick; P takes the queen of spades.

Expectation of "Flush queen of spades" plan is violated.

Apply diagnostic rules to identify problems:
Using "Invalid Premise" diagnostic rule, the derived premise in Statement 7 is falsified by the data. The inference rule used to derive this false premise is the rule specified in Statement 6. Its premise is true, but its conclusion is false. Declare this rule faulty.

Apply Learning Rules to modify plans and heuristics:
Using "Guarantee Assumed Conditions" learning rule, the system looks for other rules in the knowledge base that identify conditions for inferring Statement 7. It may find, for example, the inference rule:

If opponent player plays a high card C
and player P plays below C
then opponent wins trick and takes C

In our current example, C is the queen of spades.
This rule now replaces the faulty rule in Statement 6 with the new premise

Player P plays below the queen of spades

added as a premise to the plan and the proof. The resultant plan is

If player P takes the lead
and P doesn't have the queen of spades
then P continues leading spades
below the queen of spades

Fig. 6--Knowledge-Refinement Example

rules will lead to alternative plausible knowledge bases and associated operational procedures. Managing multiple configurations of this sort, of course, can be very difficult. Understanding how humans narrow this combinatorial space should be a principal goal of subsequent studies of human learning.

In summary, once a plan is executed, much can be learned from a retrospective analysis. During initial advice-taking, two important ingredients are missing which later can support evaluation and discovery. The first new source of information is the actual situation description. The details of the actual situation in which the plan is executed reveal and implicitly define important special cases that the general operationalization overlooks. Second, having acted, we can see the true effects of our behavior on the environment. This provides sources of confirmation or disconfirmation of parts of our plans, which then stimulate focused efforts at diagnosis and knowledge refinement. These provide numerous opportunities for concept formulation and each, in turn, initiates a new cycle of advice-taking, knowledge programming, and knowledge refinement.

CONCLUSIONS

This paper discusses

We have formulated skill development as an iterative process that converts advice into plans and, ultimately, converts these plans into behaviors. The overall model we have presented is summarized in Figure 7. While this framework treats learning as a largely domain-independent enterprise, it motivates two caveats. First, we believe every skill is

SKILL DEVELOPMENT AS AN ITERATIVE PROCESS

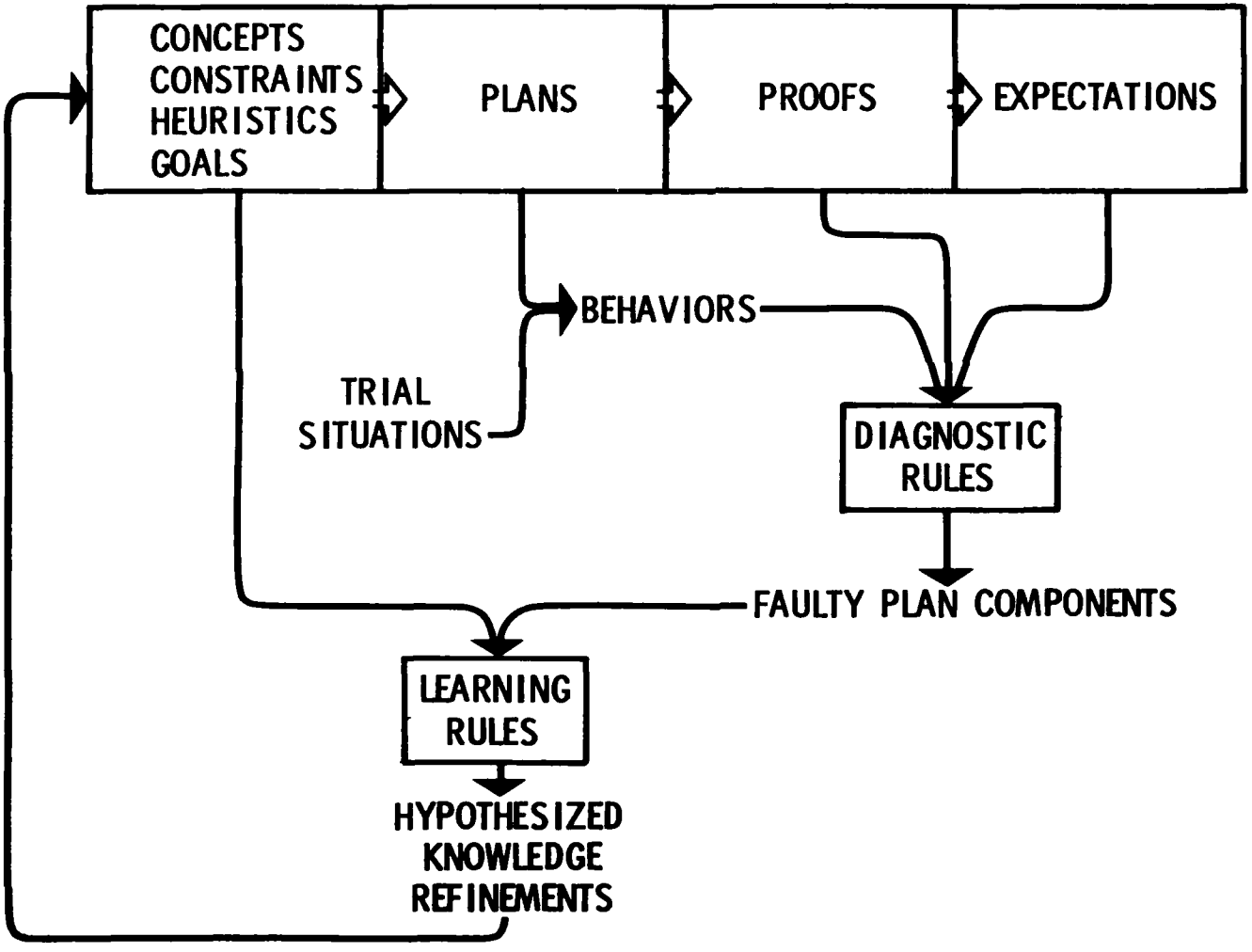


Figure 7

largely domain-dependent. Whatever domain independence exists is attributable to the general skills that underlie initial skill acquisition and subsequent skill improvement. Initial skill acquisition depends on the general and complex advice-taking skills of understanding and knowledge programming. In this paper, we have developed many aspects of the advice-taking process. The second phase of learning also employs numerous and relatively general skills. In this phase, diagnostic and learning rules identify and rectify erroneous bits of knowledge. The second caveat on domain-independence recognizes the important role that domain knowledge plays in diagnosis and refinement. A learner's ability to apply diagnosis and learning rules will also depend on his or her familiarity with and expertise in the problem domain. Although these heuristic and learning rules are domain-independent, to apply these rules a learner must be able to reason deductively about and with the entailments of his or her domain knowledge. That is, he or she must understand how specific expectations derive from particular assumptions, plans, and proofs. Furthermore, to refine erroneous knowledge, the learner must be able to reason about the knowledge and consider alternatives to it. While these are general skills, differential experience in particular problem domains will presumably affect the difficulty of learning in those domains.

We believe our model of learning processes can be translated into a practical instructional program. Humans regularly perform the functions of understanding advice, knowledge programming, diagnosis, and refinement. Students could benefit from instruction on improved methods for each of these tasks. The concepts of domain definitions, constraints,

heuristics, goals, plans, proofs, expectations, and violations should form the basis for a deeper understanding of one's own learning functions. Because these concepts make intuitive as well as analytical sense, they may support practical efforts to develop learning skills.

On the other hand, much remains to be done within the proposed learning research paradigm. While the processes we have described in this paper shed some light on advice-taking and knowledge refinement, they do not constitute a complete model. For the goals of experimental psychologists, many additional assumptions would need to be introduced to transform our current model into a more predictive and vulnerable theory. To make the model valuable in computer applications, many more of the functions need to be programmed. This is hard for two reasons. First, each of the functions we have described is a state-of-the-art objective for AI. And, if that is not enough, learning programs must cope with combinatorics arising from ambiguities inherent in advice and the alternative possibilities for operationalizing each bit of knowledge. For all these reasons, we expect only continual, incremental advances in our development of this research paradigm. In short, learning functions of the sort considered in this paper appear to yield readily to analysis; on the other hand, synthesis of learned behaviors in complex tasks will prove considerably more difficult.

REFERENCES

Allen, J. Anatomy of Lisp, New York: McGraw Hill, 1978.

Balzer, R. A mathematical model for performing a complex task in a card game. Behavioral Sciences, 1966, 2 (3), 219-236.

Balzer, R., Goldman, N., and Wile, D. Informality in program specifications. Proceedings of the Fifth International Joint Conference on Artificial Intelligence. Cambridge, Mass., 1977, 389-397.

Barr, A., Bennett, J., and Clancey, W. Transfer of expertise: A theme for AI research. Technical Report HPP-79-11, Stanford University, 1979.

Barstow, D. A knowledge-based system for automatic program construction. Proceedings of the Fifth International Joint Conference on Artificial Intelligence. Cambridge, Mass., 1977, 382-388.

Davis, R. Interactive transfer of expertise: Acquisition of new inference rules. Artificial Intelligence, 1979, 12 (2), 121-158.

Davis, R. Knowledge acquisition in rule-based systems: Knowledge about representation as a basis for system construction and maintenance. In D. A. Waterman and F. Hayes-Roth (eds.), Pattern-Directed Inference Systems, New York: Academic Press 1978, 99-134.

Hayes-Roth, F. The role of partial and best matches in knowledge systems. In D. A. Waterman and F. Hayes-Roth (eds.), Pattern Directed Inference Systems, New York: Academic Press, 1978, 557-574.

Hayes-Roth, F. Theory-driven learning: proofs and refutations for concept discovery. Rand Technical Note N-1543, Santa Monica, Calif.: The Rand Corporation, 1980.

Hayes-Roth, F., Klahr, P., Burge, J., and Mostow, D. J. Machine methods for acquiring, learning, and applying knowledge. P-6241, Santa Monica, Calif.: The Rand Corporation, October 1978.

Hayes-Roth, F., Klahr, P., and Mostow, D. J. Knowledge acquisition, knowledge programming, and knowledge refinement. R-2540-NSF, Santa Monica, Calif.: The Rand Corporation, 1980.

Heidorn, G. E. English as a very high level language for simulation programming. Proceedings of the ACM SIGPLAN Symposium on Very High Level Languages. Santa Monica, Calif.: 1974, 91-100.

Lakatos, I. Proofs and Refutations. Cambridge: Cambridge University Press, 1976.

Lenat, D. AM: An artificial intelligence approach to discovery in mathematics as heuristic search. SAIL AIM-286. Stanford, Calif.: Stanford Artificial Intelligence Laboratory, 1976. Jointly issued as Computer Science Dept. Report No. STAN-CS-76-570.

Lenat, D. and Harris, G. Designing a rule system that searches for scientific discoveries. In D. A.Waterman and F. Hayes-Roth (eds.), Pattern-Directed Inference Systems, New York: Academic Press, 1978, 25-51.

Lenat, D. B., Hayes-Roth, F., and Klahr, P. Cognitive economy, N-1185. Santa Monica, Calif.: The Rand Corporation, 1979(a).

Lenat, D. B., Hayes-Roth, F., and Klahr, P. Cognitive economy in AI systems. Proceedings of the Sixth International Joint Conference on Artificial Intelligence. Tokyo, Japan, 1979(b), 531-536.

Mostow, D. J., and Hayes-Roth, F. Machine-aided heuristic programming: A paradigm for knowledge engineering. N-1007. Santa Monica, Calif.: The Rand Corporation, 1979(a).

Mostow, D. J., and Hayes-Roth, F. Operationalizing heuristics: Some AI methods for assisting AI programming. Proceedings of the Sixth International Joint Conference on Artificial Intelligence. Tokyo, Japan, 1979(b), 601-609.

Sussman, G. J., and Stallman, R. Heuristic techniques in computer aided circuit analysis. Memo 328. Cambridge, Mass.: MIT AI Laboratory, 1975.